International Journal of Innovative Research in Computer Science & Technology (IJIRCST)
ISSN: 2347-5552, Volume 2, Issue 3, May - 2014

Comparative Study on Layout and Drawable Resource Behavior in Android for Supporting Multi Screen

Ms. Sneha Mandavia

***Abstrac*- this paper on Android deals with multi layout supports on various size of Android device. While writing an Android application developer should consider various size and type of device.**
So this paper on Android deals with certain points which developer should consider while design the layouts in Android mobile operating system.

***Index Terms- Android, PX unit, DP unit,* mdpi**

I. INTRODUCTION

Now a day various types of Android devices are available in markets which having different screen size, screen density, orientation, resolution, and Density-independent pixel (dp). As per Android documentation following is the concepts for each above terms. [1]

[1] Screen size: Actual physical size i.e. small, medium, large, and extra large.

[2] Screen density: quantity of pixels within a physical area, it referees to as dot per inch (dpi).

Four types of densities are: ldpi (low), mdpi (medium), hdpi (high), and xhdpi (extra high)

Density for Drawable i.e. Images

drawable-ldpi: for low-density (ldpi) screens (~120dpi).

drawable-mdpi: for medium-density (mdpi) screens (~160dpi). (This is the baseline density.)

drawable-hdpi: for high-density (hdpi) screens (~240dpi).

drawable-xhdpi: for extra high-density (xhdpi) screens (~320dpi).

drawable- nodpi: for all densities. These are density-independent resources. The system does not scale resources tagged with this qualifier, regardless of the current screen's density.

[3] Orientation: shall be either landscape or portrait.

Manuscript received May 05, 2014.

Ms. Sneha Mandavia, FAculty of Department of Computer Science and Technology, Uka Tarsadia University, Tarsadia. - 394 350

[4] Resolution: The total number of physical pixels on a screen.

[5] Density-independent pixel (dp): A virtual pixel unit that to use for layout dimensions or position in a density-independent way.

On a 160 dpi screen Number of Pixel = Number of density independent pixel

$$px = dp * (dpi / 160)$$

E.g. If width of control is 20dp, and application running on 240dpi screen

$$px = 20 * (240/160) = 30px$$

Types of Layouts Supported by Android:

[1] AbsoluteLayout: (This class was deprecated in API level 3.): A layout that lets you specify exact locations (x/y coordinates) of its children. Absolute layouts are less flexible and harder to maintain than other types of layouts.

[2] LinearLayout: A layout that arranges its children into rows and columns.

[3] RelativeLayout: sibling views can define their layout relative to another sibling view, which is referenced by the unique ID. You cannot have a circular dependency between the size of the RelativeLayout and the position of its children.

[4] TableLayout: A layout that arranges its children into rows and columns. A TableLayout consists of a number of TableRow objects.

[5] FrameLayout: Child views are drawn in a stack, with the most recently added child on top. The size of the FrameLayout is the size of its largest child (plus padding),

Size of Android Control:

FILL_PARENT: means that the view wants to be as big as its parent

MATCH_PARENT: means that the view wants to be as big as its parent, minus the parent's padding, if any. FILL_PARENT renamed with MATCH_PARENT science API Level 8.

WRAP_CONTENT: means that the view wants to be just large enough to fit its own internal content

Note: if developer does not want to give a control size as MATCH_PARENT or WRAP_CONTENT, then he can give absolute size also like 10px or 10dp.

To show the layout's output, I tested the same layout on different features' emulator.

Device Name	Features
A	Screen size: 240 x 320 (small) Screen density: 120 dpi Screen density type: ldpi
B	Screen size: 720 x 1280 (X- Large) Screen density: 320 dpi Screen density type: xhdpi
C	Screen size: 1280 x 800 (Large) Screen density: 160 dpi Screen density type: mdpi

II. COMPARISON BETWEEN PX AND DP UNIT

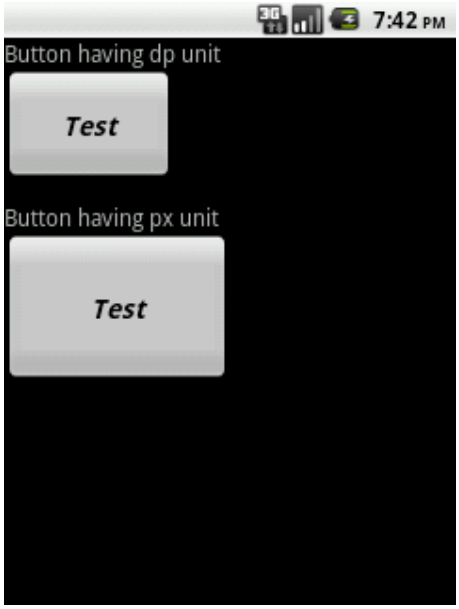
px - corresponds to actual pixels on the screen.

dp Density-independent Pixels – It is a virtual pixel. Note: The compiler accepts both "dip" and "dp".

Below example shows the difference between *dp* and *px* unit. Example having mainly two buttons, first button having hight and width in *dp* unit where as second button having hight and width in *px* unit. Below is the snippet for both controls.

```
<Button android:layout_width="120dp"
    android:layout_height="80dp"
    android:text="Test" />
<Button android:layout_width="120px"
    android:layout_height="80px"
    android:text="Test" />
```

Above snippet first tested in device – A.



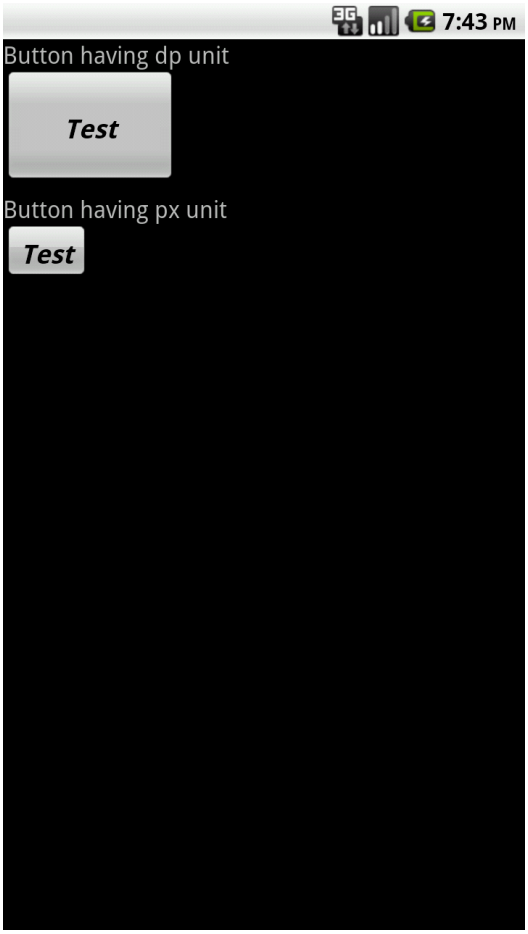
As we can see in output, the first control having size in dp, shall automatically resize with respect to device density. So as per equation control width *dp* shall converts in virtual pixel (vpx) is,

vpx = dp * (dpi / 160)
vpx = 120 * (120/160)
vpx = 90px

Control height shall,

vpx = dp * (dpi / 160)
vpx = 80 * (120/160)
vpx = 60px

Same snippet is tested in device – B.



So here again *dp* unit shall automatically resize with respect to device density. So as per equation control width *dp* shall converts in virtual pixel (vpx) is,

vpx = dp * (dpi / 160)
vpx = 120 * (320/160)
vpx = 240px

Control height shall,

vpx = dp * (dpi / 160)
vpx = 80 * (320/160)
vpx = 160px

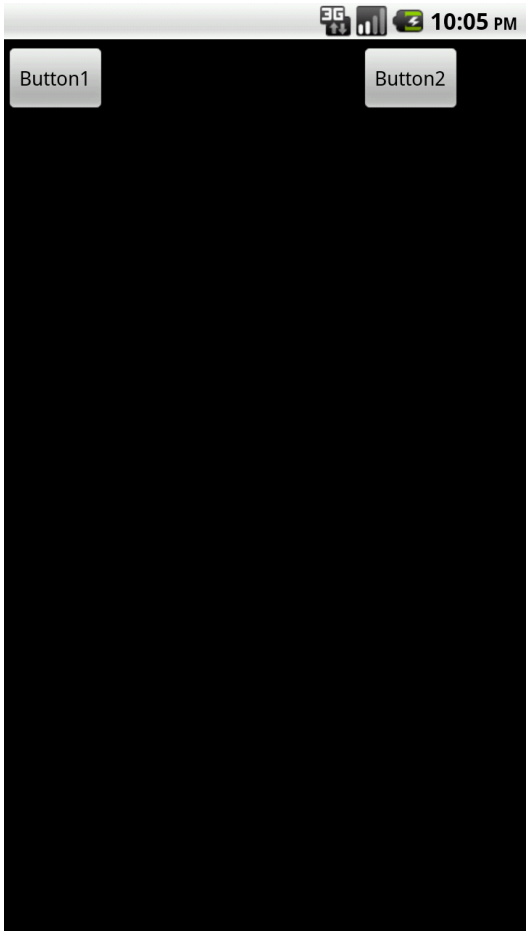
So in both the size of device, dp unit shall automatically converted to virtual pixel. But if we give px unit, remains as it is each density device.

III. PREFERRED LAYOUT TYPE

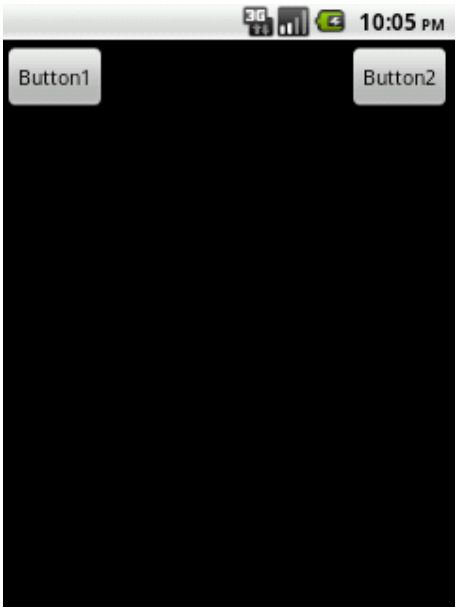
AbsoluteLayout: Below snippet shows the example for *AbsoluteLayout*. [4]

```
<AbsoluteLayout
   android:layout_width="match_parent"
   android:layout_height="match_parent">
   <Button android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_x="0dp"
      android:layout_y="5dp"
      android:text="Button1" />
   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_x="245dp"
      android:layout_y="5dp"
      android:text="Button2" />
</AbsoluteLayout>
```

Above snippet was tested in device – B.



Same snippet was tested in device - A



So here can see while using same snippet on different device, output was not proper. As control margin is absolute.

What to do if we want to execute same layout on different device? Let's try with other layout type called *RelativeLayout*.

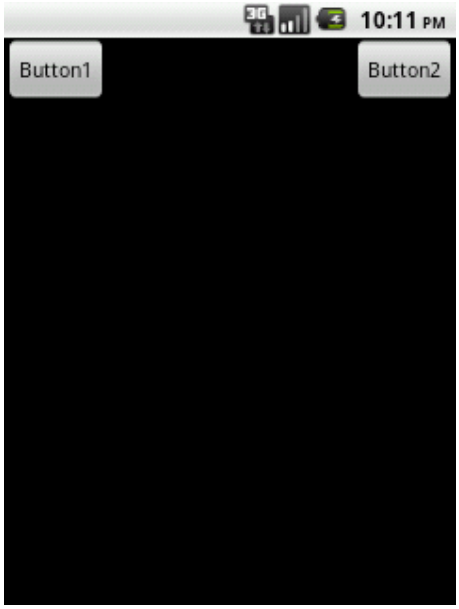
[2] **RelativeLayouts** arrange the view with related to its sibling on its parents, RelativeLayout's property:

android:layout_alignParentRight
android:layout_alignParentLeft
android:layout_alignParentTop
android:layout_alignParentBottom
android:layout_toLeftOf
android:layout_toRightOf
android:layout_above
android:layout_below

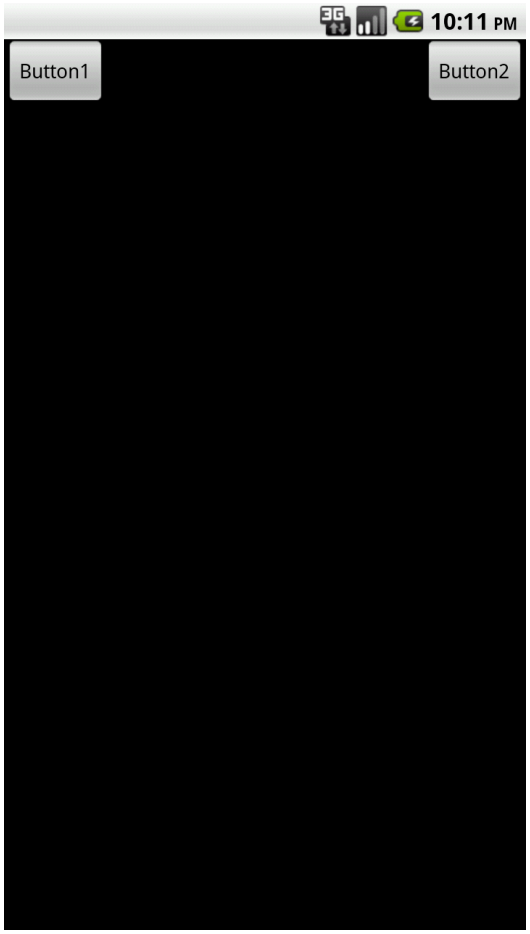
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
   android:layout_width="match_parent"
   android:layout_height="match_parent">
   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Button1" />
   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentRight="true"
      android:text="Button2" />
</RelativeLayout>
```

Above snippet sets button2 at left side of layout.

And snippet was tested in device - A.



Same snippet was tested in device – B.



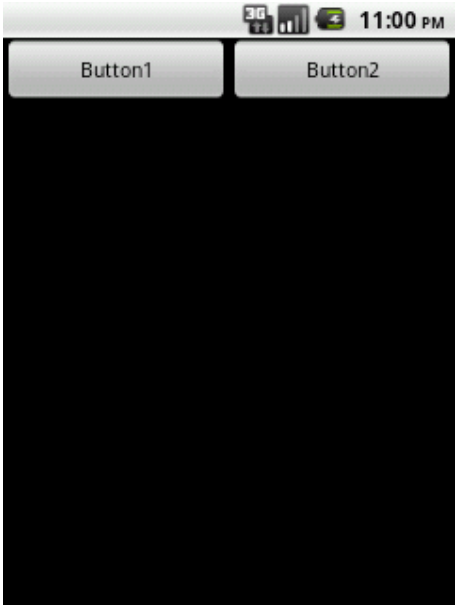
Here we can see same snippet gave same output on different density size of device.

[3] LinearLayout: name suggests, all the elements are displayed in a linear fashion, either *Horizontally* or *Vertically* and this behavior is set in android:orientation which is an attribute of the node *LinearLayout*. Another property android:layout_weight is use to give ratio for view's size.

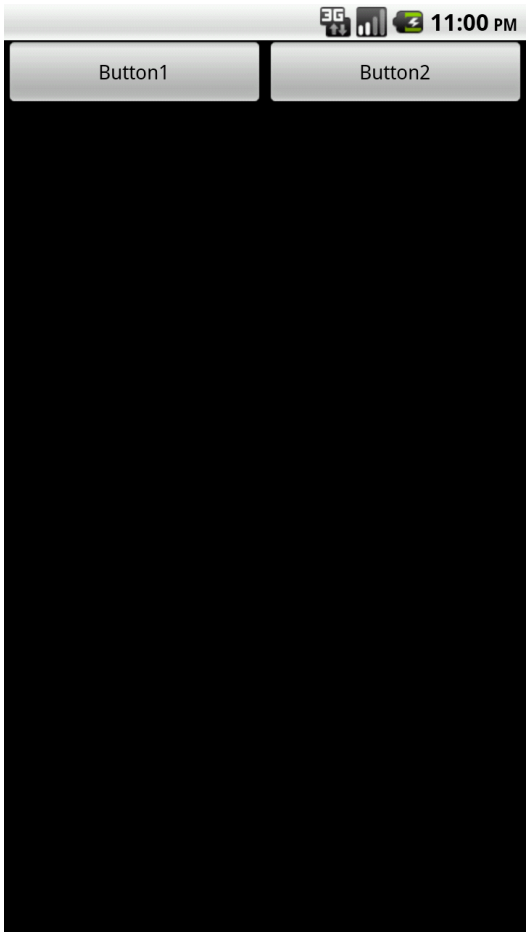
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="horizontal" >
   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_weight="1"
      android:text="Button1" />
   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_weight="1"
      android:text="Button2" />
</LinearLayout>
```

Above code was again tested in two types of device, and it's seems that on both the device, layout's output is accordingly screen size and screen density.

Snippet was tested in device – A.



Snippet was tested in device- B.



So I both the size of device LinearLayout shows same output based on device size and density.

IV. USING DRAWABLE RESOURCE

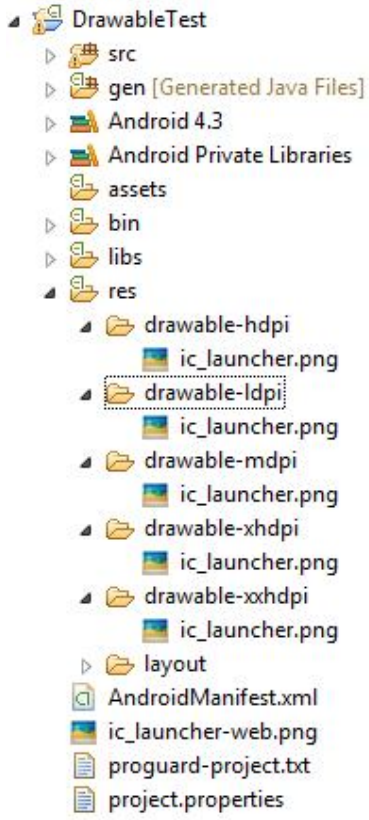
d*p* unit is use to set control height and width, but if wants to set background resource, then we have to use drawable resource for a graphic that can be drawn to the screen.[5]

Here I have shown the Android application structure having different drawable folder. So if developer wants to displays the image recourse as good as on any screen size having any density, he has to put a same image having different size but same name in different folder. So whenever user install same application on any device, based on device size and density application shall automatically pick the image from corresponding folder.

Example: If I am executing my application in medium density device, so application shall pick the image from drawalbe-mdpi.

Folder specification:

Folder name	density	Screen size	Image size
drawable-ldpi	120(low)	Small	0.75 x mdpi
drawable-mdpi	160(medium)	Medium	Base size
drawable-hdpi	240(large)	Large	1.50 x mdpi
drawable-xhdpi	320(extra large)	Extra large	2 x mdpi

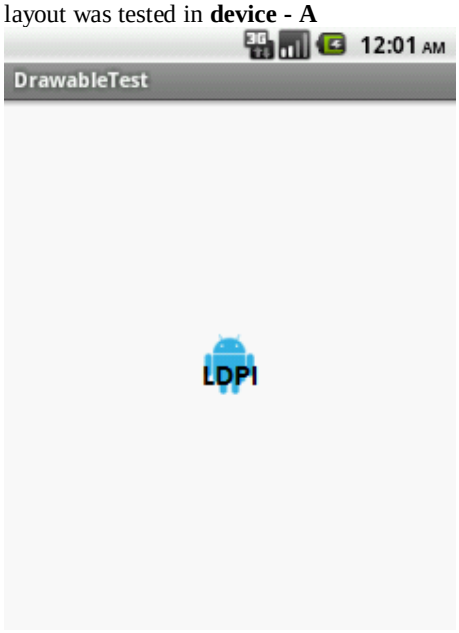


So below snippet was tested on different screen size.

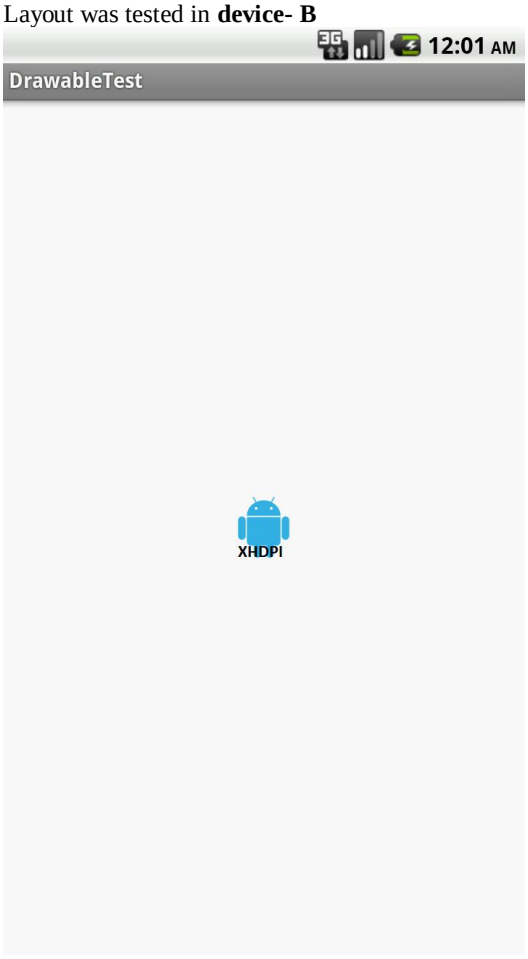
```
<RelativeLayout
  xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent" >

   <ImageView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_centerInParent="true"
      ndroid:background="@drawable/ic_launcher" />
</RelativeLayout>
```

layout was tested in **device - A**



Layout was tested in **device- B**

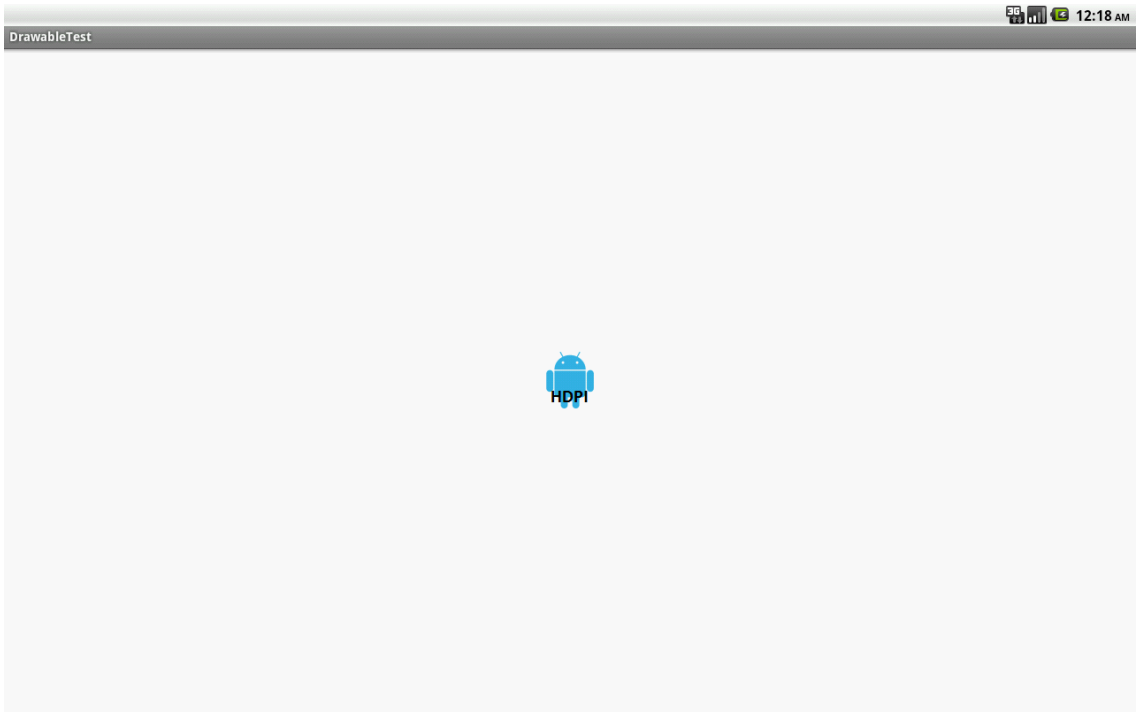


Layout was tested in **device - C**

Output comparison on different screen size,

Device Name	Features			Image pick from
	Screen size	Screen density	Screen density type	
A	240 x 320 (small)	120 dpi	ldpi	drawable-ldpi
B	720 x 1280 (X- Large)	320 dpi	xhdpi	drawable-xhdpi
C	1280 x 800 (Large)	160 dpi	mdpi	drawable-mdpi

While executing the application device- c, Android picks the image from drawable-mdpi folder, as per device density device picks image from the right folder, but device size is large. So it's looks not good, as Android picks medium size image for large size device. So to remove this deficiency we have to add new resource i.e. creates a new folder called **drawable-large-mdpi. drawable-large-mdpi:** resource is use for the device which have medium density but large screen size, so we have to copy all the image from drawalbe-hdpi folder, and paste them to drawable-large-mdpi, so as a result we achieves the following output.



So In this way developer can make n-number of resource for different screen size of device, whose density shall not match with the provided drawable.

As per convention developer should create all size of images for each folder, but if some time developer avoid or miss to put any image in a folder, then Anroid is enough smart. It picks the image from another drawable folder. For example if developer made images for drwable-ldpi folder, and then he tries to execute his application on extra large screen, then extra large screen device picks the image from drwable-ldpi. But at that time on large screen output shall not as much good as low screen size device as drawable quality shall lose.

V. USING SP UNIT

Scale-independent Pixels - this is like the dp unit, but it is also scaled by the user's font size preference. It is recommend you use this unit when specifying font sizes, so they will be adjusted for both the screen density and user's preference. [6]

VI. CONCLUSION

[1] When developer requires to give control's size, always prefer *dp* unit rather than *px* unit.

[2] Avoid to use *AbsoluteLayout* while designs the layout. *RelativeLayout* is a great way to go. If you set Views' distances and sizes with dp, those distances in dp will be treated as pixels on an MDPI screen i.e. 160dpi screen, and they will be scaled so your layouts fill the space equally on LDPI and HDPI screens.

[3] If possible create all size of drawable for all screen size and screen density device, so in future your application looks good on any size of device.

VII. REFERENCES

[1]http://developer.android.com/guide/practices/screens_support.html
[2]http://developer.android.com/guide/topics/ui/declaring-layout.html
[3]http://stackoverflow.com/questions/16708078/what-is-difference-between-photoshop-px-and-android-dp
[4]http://www.androidhive.info/2011/07/android-layouts-linear-layout-relative-layout-and-table-layout/
[5]developer.android.com/guide/topics/resources/drawable-resource.html
[6]http://developer.android.com/guide/topics/resources/more-resources.html#Dimension

Ms. Sneha Mandavia is a faculty of department of Computer Science and Technology, Uka Tarsadia University, Tarsadia. - 394 350. She is working as teaching assistant at same institute since last two years and teaching different subject like Mobile Application Development (Android), Web Application Development (J2EE), Object Oriented Software Engineering, Computer Graphics, Java Programming. She has one year experience in Android mobile application development. She has completed M.Sc. (IT) from J.P. Dawer Institute of Information Science & Technology, VNSGU, Surat.